PROJECT 1

Ερώτηση 1.

Για την αναζήτηση πρώτα σε βάθος θα χρησιμοποιήσουμε μια LIFO ουρά που είναι υλοποιημένη στη κλάση Stack στο utils.py αρχείο. Ακόμα, επειδή ζητείται η αναζήτηση να γίνεται σε γράφο θα χρησιμοποιήσουμε σύνολο (set) για την διαχείριση του συνόρου.

Οι κόμβοι αναπαρίστανται ως πλειάδες που αποτελούνται από την κατάσταση (θέση προβλήματος, state) και τη διαδρομή (path) απο τη ρίζα μέχρι τη κατάσταση που περιγράφει ο κόμβου (path -> λίστα απο ενέργειες που απο τη ρίζα πάνε στον κόμβο)

Ερώτηση 2.

Η διαφορά οφείλεται στον ίδιο λόγο για τον οποίο στον “dfs” χρησιμοποιούμε ουρά “LIFO” ενώ στον “bfs” ουρά “FIFO” κατά την υλοποίησή τους. Ο “dfs” διερευνά πρώτα σε βάθος τους κόμβους (οπότε οι κόμβοι που βρίσκονται κοντά στη ριζά θα διερευνηθούν αργότερα από τους κόμβους ενός εκ των παιδιών της ρίζας για ‘ αυτό “ first in” -> κοντά στη ρίζα οπότε μπαίνει στο σύνορο νωρίς “last out” -> βρίσκεται σε μεγαλύτερο βάθος/ τα φύλλα πρώτα οπότε μπαίνει στο σύνορο αργότερα αλλά διερευνάται γρήγορα), οπότε συνήθως θα μελετάει λιγότερους κόμβους ώσπου να βρει λύση που όμως πιθανόν να μην είναι βέλτιστη. Αυτό διότι λύση για το πρόβλημα μπορεί να υπάρχει και στο τελευταίο παιδί της ρίζας (σε ένα από τα παιδιά του παιδιού)

Από την άλλη μεριά ο “bfs” διερευνά όλους τους κόμβους ανά επίπεδο (οι κόμβοι όσο πιο κοντά στη ρίζα είναι τόσο πιο γρήγορα θα διερευνηθούν γι ‘ αυτό “first in” -> κοντά στη ρίζα άρα μπαίνει γρήγορα στο σύνορο “first out” -> όλοι οι κόμβοι του ίδιου επιπέδου και είναι κοντά στη ρίζα που είναι μέσα στο σύνορο θα διερευνηθούν πρώτοι) , οπότε συνήθως θα χρειαστεί να διερευνήσει όλους τους κόμβους εώς και την λύση . Η λύση που θα βρεί είναι σίγουρο πως θα είναι βέλτιστη ως προς το ύψος διότι διερευνά κάθε κόμβο κάθε επιπέδου ξεκινώντας από τη ρίζα προς τα φύλλα.

Ερώτηση 3.

Το pos είναι tuple ή λίστα και το pos[0] συγκεκριμένα είναι η τετμημένη χ.

Η συνάρτηση κόστους του πράκτορα “StayEastsearchAgent” είναι “ costFn = lambda pos: .5 ** pos[0] “. Στα μαθηματικά ο τύπος είναι : $\frac{1}{2^x}$, οπότε βλέπουμε πως καθώς ο πράκτορας πηγαίνει ανατολικά (δεξιά) αυξάνεται η τετμημένη άρα μεγαλώνει ο παρανομαστής και ακολούθως μειώνεται το κόστος costFn ενώ όταν πηγαίνει δυτικά (αριστερά) μειώνεται ο παρονομαστής με μικρότερη τετμημένη και άρα αυξάνεται το κόστος costFn. Έτσι, προτιμάται ο πράκτορας να είναι όσο πιο ανατολικά γίνεται όπου το κόστος είναι ελάχιστο.

Η συνάρτηση κόστους του πράκτορα “StayWestsearchAgent”

” είναι “ costFn = lambda pos: .5 ** pos[0] “. Στα μαθηματικά ο τύπος είναι : 2^{χ}, οπότε βλέπουμε πως καθώς ο πράκτορας πηγαίνει δυτικά (αριστερά) μειώνεται η τετμημένη άρα μειώνεται το κόστος costFn ενώ όταν πηγαίνει ανατολικά (δεξιά) αυξάνεται η τετμημένη χ άρα και το κόστος costFn. Έτσι, προτιμάται ο πράκτορας να είναι όσο πιο δυτικά γίνεται όπου το κόστος είναι ελάχιστο.

Ερώτηση 4.

Ο A* χρειάστηκε να διευρύνει 549 κόμβους ενώ ο UCS απαιτεί 620 κόμβους στον bigMaze.

Ο A* χρειάστηκε να διευρύνει 535 κόμβους ενώ ο UCS απαιτεί 682 κόμβους στον openMaze.

Ερώτηση 5.

Λογική της ευρετικής: Καταγράφει τις θέσεις των τροφών και ελέγχει αν υπάρχει τροφή. Στη συνέχεια, υπολογίζει την μέση Manhattan απόσταση του pacman από τις τροφές.

Η απόσταση αυτή είναι η ευρετική τιμή.

Επεκτείνονται 11254 κόμβοι.

Αποδεκτότητα: Η μέση απόσταση είναι πάντοτε ίση ή μικρότερη από την πραγματική απόσταση που πρέπει να διανύσει ο Pacman για να συλλέξει όλα τα τρόφιμα, κάνοντας την ευρετική αποδεκτή.

Συνέπεια: Η μέση απόσταση είναι συνεπής επειδή οι αποστάσεις που υπολογίζονται είναι Μανχάταν αποστάσεις και ακολουθούν την τριγωνική ανισότητα.

Ερώτηση 6.

Πως εξηγείτε τη συμπεριφορά του Pacman στην εκτέλεση του παιχνιδιού q6 παραπάνω;

Κατά την εκτέλεση, ο αλγόριθμος αξιολογεί όλες τις πιθανές κινήσεις που μπορεί να κάνει ο Pacman και τους φαντάσματες, και επιλέγει αυτή που του εξασφαλίζει τη μέγιστη πιθανή αξία (σύμφωνα με τη λογική του Minimax). Εάν είναι η σειρά του Pacman να παίξει (max), επιλέγει την ενέργεια που οδηγεί στη μέγιστη πιθανή αξία (βαθμολογία) για τον Pacman. Αντίθετα, αν είναι η σειρά των φαντασμάτων να παίξουν (min), επιλέγει την ενέργεια που οδηγεί στην ελάχιστη πιθανή αξία για τον Pacman (δηλαδή, τη μέγιστη αξία για τα φαντάσματα).

Έτσι, ο Pacman προσπαθεί να παίξει με τρόπο που θα του εξασφαλίσει τη μέγιστη δυνατή βαθμολογία, λαμβάνοντας υπόψη τις ενέργειες και τις αποφάσεις των φαντασμάτων μέσω του αλγορίθμου Minimax.

Γιατί ενώ εκτελούμε τον αλγόριθμο MiniMax ο πράκτοράς μας μπορεί να χάνει;

Διότι ίσως να μην έχει αναλύσει σε αρκετό βάθος το δέντρο και έτσι να καταλήξει σε μια κατάσταση πού δεν μπορεί να κερδίσει.

Γιατί ο πράκτορας σε ορισμένες περιπτώσεις περιμένει τα φαντάσματα να πλησιάσουν;

Επειδή ίσως να είναι στρατηγικά καλύτερο να περιμένει τα φαντάσματα να πάνε σε συγκεκριμένες θέσεις και ύστερα να αλλάξει θέση. Να έχει περισσότερες πιθανότητες να κερδίσει.

Ερώτηση 7.

Εξηγήστε γιατί συμβαίνει αυτό στην περίπτωση του MinimaxAgent .
Πότε είναι λάθος η συγκεκριμένη στρατηγική και γιατί;

Ίσως οφείλεται στο ότι προσπαθεί να ελαχιστοποιήσει το χρόνο κατά τον οποίο βρίσκεται σε losing position διότι χάνει πόντους όσο βρίσκεται σε losing position, οπότε προτιμά να χάσει πιο γρήγορα.

Μπορεί να είναι λάθος όταν μέσα στους στόχους του παιχνιδιού είναι και το να επιβιώσει για περισσότερο χρόνο. Ακόμη, αν τα φαντάσματα/πράκτορες γνωρίζουν τη τακτική ελαχιστοποίησης βημάτων του pacman τότε ίσως το εκμεταλλευτούν.

Ερώτηση 8.

Η συνάρτηση αξιολόγησης betterEvaluationFunction παίρνει την τρέχουσα κατάσταση του παιχνιδιού ως είσοδο και επιστρέφει μια αξιολόγηση για αυτήν την κατάσταση. Η λειτουργία της συνάρτησης είναι να εκτιμήσει πόσο καλή είναι μια δεδομένη κατάσταση του παιχνιδιού για τον παίκτη (τον πακμαν) με βάση διάφορα κριτήρια.

Η συνάρτηση αξιολόγησης χρησιμοποιεί τα παρακάτω στοιχεία της τρέχουσας κατάστασης του παιχνιδιού:

- Θέση του πακμαν
- Λίστα τροφίμων που πρέπει να φάει ο πακμαν
- Κατάσταση των φαντασμάτων (θέσεις και χρόνοι που είναι φοβισμένα)
- Κάψουλες που υπάρχουν στο παιχνίδι

Η συνάρτηση αξιολόγησης υπολογίζει ένα σκορ για την τρέχουσα κατάσταση με βάση τα παραπάνω στοιχεία και τα εξής κριτήρια:

1. Απόσταση από το πλησιέστερο φαγητό: Προσθέτει την αντίστροφη της απόστασης από το πλησιέστερο φαγητό στο σκορ.
2. Απόσταση από το πλησιέστερο φάντασμα: Αν το φάντασμα είναι φοβισμένο, προσθέτει την αντίστροφη της απόστασης από το φάντασμα, αλλιώς αφαιρεί μια τιμή βάσης από το σκορ.
3. Απόσταση από την πλησιέστερη κάψουλα: Προσθέτει την αντίστροφη της απόστασης από την κοντινότερη κάψουλα στο σκορ.
4. Μείωση σκορ για τον αριθμό των φαγητών που απομένουν.
5. Μείωση σκορ για τον αριθμό των κάψουλων που απομένουν.

Τα βάρη που χρησιμοποιούνται για κάθε κριτήριο μπορεί να προσαρμοστούν ανάλογα με την επίδοση του πακμαν. Η συνάρτηση αξιολόγησης επιστρέφει το τελικό σκορ.

Θεωρώ πως η διαφορά στις συμπεριφορές οφείλεται στην στρατηγική χρήση των καψουλών και στις κινήσεις που γίνονται όταν τα φαντάσματα είναι φοβισμένα.
Ο minimax δεν φαίνεται να λαμβάνει υπόψη του την επιπλέον ικανότητα που δίνει η κάψουλα και της συμπεριφέρεται σαν να είναι απλή τροφή και επιπλέον μοιάζει να μην γνωρίζει ότι μπορει να φάει τα φοβισμένα φαντάσματα.
Σε αντίθεση, ο πράκτορας στο ερώτημα 8 υπολογίζει και προσπαθεί να εκμεταλλευτεί και αυτούς τους 2 παράγοντες.

Σημείωση:
Χρησιμοποιήθηκε βοήθεια από chatgpt,
wikipedia(https://en.wikipedia.org/wiki/Minimax ,
https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning ,
https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm#Practical_optimizations_and_infinite_graphs , https://en.wikipedia.org/wiki/Depth-first_search ,
https://en.wikipedia.org/wiki/Breadth-first_search ,
https://en.wikipedia.org/wiki/A*_search_algorithm),
github(https://github.com/PetropoulakisPanagiotis/pacman-projects/tree/master , https://github.com/karlapalem/UC-Berkeley-AI-Pacman-Project/tree/master , https://github.com/pspanoudakis/Berkeley-Pacman-Projects/tree/master)